Isolation Forest

> F. T. Liu, K. M. Ting and Z. Zhou, "Isolation Forest," 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 413-422, doi: 10.1109/ICDM.2008.17.

Reviewer: 박이현, 이진모, 홍현경

Abstract

현재 존재하는 모델 기반의 이상치 탐지 알고리즘은 정상치의 특징을 구성(profile construct)한 후 그 특징에 위배되는 경우를 이상치로 분류하는 형태로 작동한다. 논문에서 제시된 Isolation Forest는 이러한 기존의 방법론을 따르지 않고, 정상치의 특징을 구성하지 않으면서 이상치를 정상치로부터 격리하는 알고리즘이다. 부분 샘플링을 활용하기 때문에 선형의 시간 복잡도를 달성하며 메모리 소모 역시 적다.

1. Introduction

이상치란 정상치들과 상이한 성격을 띠는 데이터 패턴을 의미한다. 이러한 이상치를 탐지해내는 것은 일상의 다양한 영역에서 유용하게 쓰인다. 신용카드 거래 내역에서의 이상치는 거래 사기를 의미할 수도 있고 네트워크 트래픽 기록에서의 이상치는 허용되지 않은 접근을 의미할 수 있다.

따라서 이상치를 탐지해내는 여러가지 모델 기반의 알고리즘이 개발되었는데, 그 작동방식은 초록에서 설명한 바와 같고, statistical method, classification-based method, 그리고 clustering-based method 등이 있다. 그러나 이러한 알고리즘에는 다음 2가지의 결함이 존재한다.

- 정상치의 특징을 구성하기 때문에 이상치를 찾아내는데 최적화되지 못한다. 그 결과 정상치를 이상치로 오분류하거나 찾아내는 이상치가 지나치게 적은 문제가 발생한다.
- 거리나 밀도 계산 등을 요하기 때문에 계산 capacity와 메모리 소요 때문에 저차원의 적은 데이터에 대해서만 작동한다.

논문에서 제시하는 이상치 탐지 알고리즘은 이러한 정상치에 대한 프로파일링을 거치지 않는다. 다만,

1) 이상치는 그 수가 적고
2) 정상치들과는 상이한 value를 갖는다는 점을 활용한다.

1과 2를 종합해서 사고해보면, 2차원 평면상에 흩뿌려진 데이터 포인트들을 선을 그어 하나씩 이격시킬 때, 이상치일수록 더 적은 선만 가지고 이격시킬 수 있다 (1부터 25, 그리고 50으로 이루어진 숫자 포인트가 흩뿌려져 있을 때 16을 찾는 것보다 50을 찾는 데 시간이 더 적게 걸리는 것처럼). 이러한 이상치의 특징을 '민감도(susceptibility)'라고 표현하겠다. 따라서 Tree의 형태로 데이터 포인트의 값을 기준으로 그들을 분할해 나가면, 이상치들은 민감도 때문에 적은 횟수의 분할만으로도 이격이 되고, 반대로 정상치들은 여러 번 분할해야만 이격된다. **즉, 이상치일수록 root node와 가깝다 (= 분할 횟수가 적다)**.

위에서 설명한 개념으로 이상치를 탐지해내는 모델이 논문에서 제시하는 Isolation Forest, 짧게 iForest이다. 여러 개의 Tree를 앙상블한 형태이기 때문에 여러가지 Tree들을 종합해 짧은 평균 분할 거리를 갖는 샘플들을 이상치로 분류한다. 이러한 iForest가 기존의 거리 기반 혹은 밀도 기반 알고리즘과 차별되는 점은 다음과 같다.

- 개별 iTree들이 끝까지 분할될 필요가 없다. 이상치를 찾았으면 정상치를 모두 이격시킬 필요가 없기 때문이다. 즉, Partial model을 만들어낼 수 있다. 또한 부분 샘플링을 사용한다는 점도 기존의 모델에서는 발견되지 않는 특징이다.
- 거리나 밀도 계산이 전혀 이루어지지 않기 때문에 컴퓨팅 측면에서의 소모가 적다.
- 선형의 시간 복잡도를 달성한다. 시간 복잡도는 데이터의 양에 따라 주어진 task를 달성하는 데 필요한 시간을 의미하는데, 그 관계가 선형이라는 것은 오로지 데이터의 양만이 처리 시간에 영향을 줌을 의미한다. 논문 발표 당시 가장 최신의 알고리즘조차 선형의 시간 복잡도에 도달하지 못했다.
- iForest는 대용량 데이터 또는 고차원 데이터에 대해서도 잘 작동한다.

지금까지 설명한 알고리즘에 대한 기본적인 이해를 바탕으로, 본 논문에서는 구체적인 작동원리, 오분류에 있어서 위 작동원리가 갖는 이점, 실제 작동 알고리즘, 그리고 기존 알고리즘과의 다양한 데이터셋에 대한 성능 비교를 다룬다.

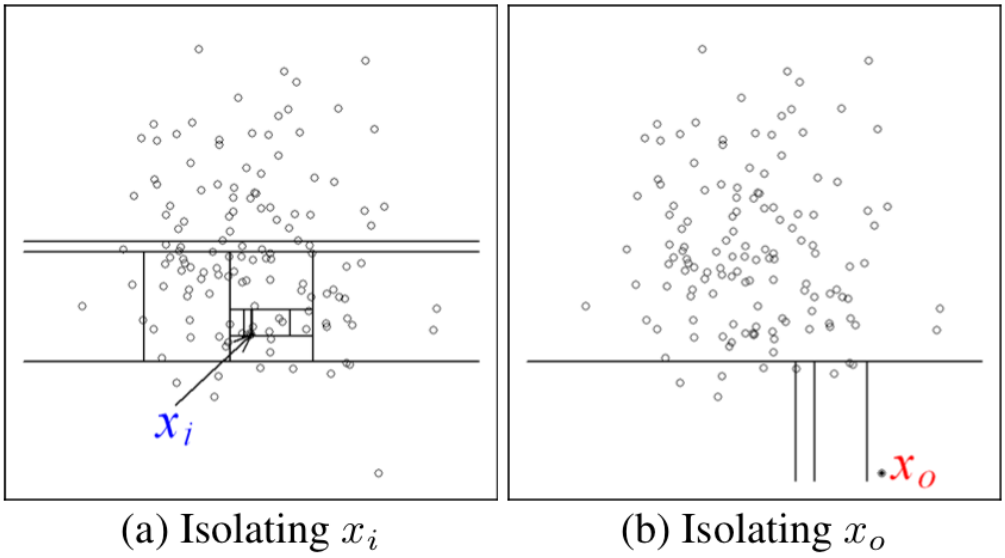
2. Isolation and Isolation Trees

반복적으로 등장하는 '이격'이라는 단어는 '하나의 샘플을 나머지 샘플들로부터 떨어트린다'는 의미이다. 이상치들은 정상치보다 '그 수가 적고 다르기' 때문에 이격에 있어서 '민감'하다. 개별 iTree들은 모든 데이터가 각각 이격될 때까지 분할된다. 이러한 분할 과정에서 이상치들은 훨씬 짧은 분할 거리(적은 분할 횟수)를 보일 수밖에 없다. 그 이유는 기본적으로 적

은 개수일수록 적은 횟수만으로 분할이 가능하고, 또한 정상치들과 차별되는 value를 가질수록 분할 초기에 이격되기 때문이다.

따라서 여러 개의 랜덤한 iTree들이 모여 iForest를 구성했을 때, 절대적으로 **짧은 평균 분할 거리(횟수)**를 보이는 포인트라면 이상치일 가능성이 커지는 셈이다.

이상치일수록 분할에 민감하다는 아이디어를 뒷받침하기 위해 논문에서는 예시를 든다.

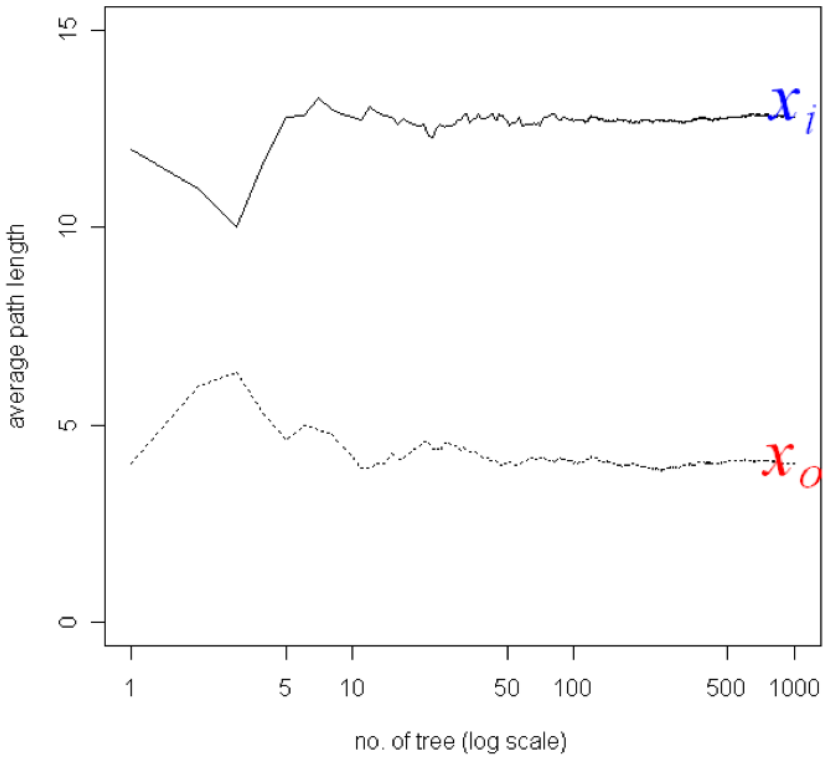


(a) Isolating x_i (b) Isolating x_o

사진 출처: Fei Tony Liu et al., 2008

위 그림에서 x_i는 정상치, x_0는 이상치이다. 위 그림에서와 같은 반복적인 분할 프로세스가 즉 Tree 형태의 구조를 의미한다는 것은 자명하다. 따라서 그림에서의 분할 횟수는 Tree에서의 분할 거리를 의미한다. 위 예시에서 분할의 기준(test)은 데이터 value의 minimum과 maximum 사이 임의의 값으로 정했다. 그림으로부터 이상치가 정상치에 비해 훨씬 적은 횟수만으로도 이격되었음을 알 수 있다. 즉 분할에 민감하다.

각각의 분할 과정이 임의로 진행되었기 때문에(기준이 random하게 정해지기 때문에) 개별 Tree들은 모두 각자 다른 데이터에 대해(부분 샘플링 하기 때문) 각자 다른 형태로 분할을 진행한다. 그렇기 때문에 각 Tree에서의 분할 횟수의 합 등이 아닌 **평균 분할 거리**를 구하여 그로부터 이상치를 판단하는 것이다. 논문에서는 위 예시에 대해 1000 개의 iTree를 만들어 그로부터 empirical하게 각 데이터 포인트에 대한 평균 분할 거리를 계산했다.



(c) Average path lengths converge

사진 출처: Fei Tony Liu et al., 2008

그 결과 이상치인 x_0의 평균 분할 거리는 4.02로 수렴했고, 정상치인 x_i의 평균 분할 거리는 12.82로 수렴하며 이상치의 평균 분할 거리가 empirical하게 정상치의 것보다 짧음을 보였다.

iTree에 대한 정의는 다음과 같다.

> T를 iTree의 node라고 할 때, 이 때 T는 하위 node가 없는 external-node이거나, 하나의 조건에 의해 정확히 2개의 하위 node(T_l, T_r)로 나뉘는 internal-node일 수 있다. 이 때 '조건'은 분할 기준 변수인 q와 그 기준 변수의 값인 p로 이루어진다. 즉, $q < p$인 경우 T_l로 분할되고, 그렇지 않으면 T_r로 분할된다.

이에 근거하여 iTree의 생성 원리를 설명해보자.

주어진 데이터 샘플 $X = \{x_1, \cdots, x_n\}$에 대하여 iTree를 만들기 위해 우선 랜덤한 기준 변수 q와 그 변수의 분할 기준 값 q를 두고이를 반복적으로 랜덤하게 지정하며 X를 분할한다. 이 때 분할은 1) Tree의 길이가 사전에 지정한 제한 길이에 도달하거나, 2) $|X| = 1$을 만족하거나, 3) 분할된 X의 값이 모두 동일해질 때까지 지속된다. iTree는 완벽하게 binary tree의 형태를 갖는다. 즉 모든 node는 하위 노드를 아예 갖지 않거나 정확히 2개의 하위 노드만을 가질 수 있다. 따라서 모든 포인트가 distinct할 경우 끝까지 자란 iTree는 각각의 포인트가 각각의 external node에 배치된 형태를 띤다. 이 경우 external node의 개수는 n개이고, internal node의 개수는 n-1개 이므로 전체 node 개수는 2n-1이고, 이에 따라서 iTree를 만드는 데 소요되는 메모리 사용량 역시 n의 크기에 따라 선형으로 증가할 수밖에 없다.

그렇다면 iTree를 활용하여 어떻게 이상치를 이상치라고 분류할 수 있는 것일까? 이상치의 경우 이격되기까지의 분할 횟수가 적음을 기억해보면, 분할 횟수가 적은 순서대로 데이터 포인트를 정렬하면 상위에 있는 데이터 포인트들을 이상치라고 판단할 수 있다. 반복적으로 등장하는 용어인 각 데이터 포인트 x의 분할 횟수(=분할 거리)는 $h(x)$로 표현하고, 그 값은 x가 iTree에 의해 완전히 이격될 때까지 통과한 node의 개수를 세어 계산한다.

그러나 하나의 $h(x)$만으로 어떤 데이터를 이상치라고 판단할 수는 없는데, 왜냐하면 iForest는 여러 개 iTree의 앙상블로서 작동하기 때문이다. 따라서 $h(x)$로부터 적절하게 도출할 수 있는 이상치로서의 기준이 필요한데, 이를 Anomaly score라고 한다. 앞서서 표현한 **평균 분할 거리**가 바로 anomaly score인 셈이다. 만들어지는 iTree들은 앞서 말했듯이 완벽하게 binary search tree(BST)의 형태를 갖는다. 그렇기 때문에 iTree에서 external node termination된 경우에 대한 $h(x)$의 평균의 추정치는 BST에서 unsuccessful search인 case에서 분할 거리 평균의 추정치로 사용하는 term을 차용해 계산할 수 있다. Unsuccessful search에 대한 설명을 아래 참고에 간단하게 추가한다.

▼ **Unsuccessful Search in BST?**

BST에서 successful search란, 분할이 완전히 이루어지기 전에 원하는 정답을 찾는 경우를 말한다. 이 경우 탐색은 internal node에서 완료된다. 그러나 internal node에서 탐색을 완료할 수 없는 경우, 즉 원하는 정답을 못 찾은 경우 탐색은 unsuccessful case가 되고 BST는 external node까지 모두 분할되어야 terminate 된다.

Unsuccessful search in BST에서 분할 거리의 평균은 아래 수식으로 계산된다.

$$c(n) = 2H(n-1) - (2(n-1)/n)$$

이 때 $H(n)$은 $ln(n) + \gamma$ (오일러 상수)로 계산된다. 위 수식의 유도 과정은 자료구조 및 알고리즘, 그리고 복잡도에 대한 개념을 필요로 하므로 유도는 생략한다. 다만 후술할 참고

자료에 추가로 공부할 수 있는 자료를 기재하겠다.

이 때 $c(n)$은 데이터 샘플의 개수 n이 주어졌을 때 $h(x)$의 평균이므로, 이를 활용해 $h(x)$를 정규화 할 수 있다. 이로부터 Anomaly score가 도출되고, 그 식은 아래와 같다.

$$Anomaly\ Score\ s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

위 식에 따라 개별 iTree의 분할 거리의 평균인 $E(h(x))$와 그에 따른 anomaly score의 관계는 아래처럼 정리된다.

- $when\ E(h(x))\ \rightarrow\ c(n),\ s \rightarrow 0.5$
- $when\ E(h(x))\ \rightarrow\ 0,\ s \rightarrow 1$
- $when\ E(h(x))\ \rightarrow\ n-1,\ s \rightarrow 0$

위의 관계를 그래프로 표현하면 아래와 같다.

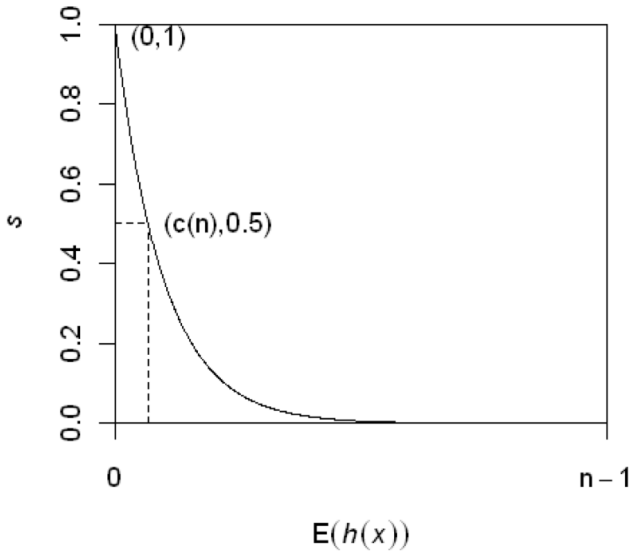


사진 출처: Fei Tony Liu et al., 2008

이를 토대로 anomaly score s를 기준으로 하는 이상치 탐지는 아래처럼 진행된다.

- 어떤 데이터 포인트의 s가 1에 근사할 경우 이상치이다.
- 어떤 데이터 포인트의 s가 0.5보다 작다면 정상치로 보기에 충분하다.
- 모든 데이터 포인트의 s가 0.5에 근사한다면 이상치가 없는 데이터로 볼 수 있다.

▼ 참고

3. Characteristics of Isolation Trees

트리 기반의 앙상블 모델인 iForest는 path length가 짧은 점들을 이상치로 탐지한다. 이 때, iForest가 모든 정상치를 격리해낼 필요가 없기 때문에 샘플링을 이용하게 된다.

기존의 모델들은 보통 샘플 사이즈가 클수록 좋은 성능을 낸다. 그러나, iForest의 경우 샘플 사이즈가 크게 되면 이상치를 탐지하는 능력이 떨어지게 된다. 즉, iForest는 기존 모델과는 달리 샘플 사이즈가 작은 경우에 더 좋은 성능을 낸다. 이러한 이유에서, iForest는 샘플을 비복원추출하는 방법을 이용한다.

이러한 비복원추출의 방법은 swamping과 masking 문제를 해결하는데 도움이 된다. 이상치 탐지 분야에서, swamping은 정상치를 이상치로 잘못 탐지해 내는 것을 의미하고, masking은 다수의 이상치가 존재하여 이상치가 탐지되지 않는 것을 의미한다. 두 경우 모두 이상치를 탐지해내기 위한 분할의 수가 증가하기 때문에 path length 역시 증가하게 되고, 이로 인하여 이상치 탐지가 어렵게 된다.

Swamping과 masking은 이상치 탐지를 위한 데이터가 너무 많을 때 발생하는 문제이다. 그런데, 앞서 언급한 바와 같이 iForest는 샘플을 비복원추출하여 partial model을 만든다. 비복원추출을 시행하게 되면 데이터 사이즈가 줄어들기 때문에 이상치를 더 잘 탐지해낼 수 있게 된다. 더불어, 비복원추출을 한 데이터를 통해 하나의 트리를 만들게 되면 서로 다른 이상치 데이터가 각 트리를 만들기 위한 입력값으로 들어가거나 입력값에 이상치가 존재하지 않을 수 있으므로 각 트리가 이상치를 찾는데 더 전문화 될 수 있다. 이러한 특징으로 인하여 iForest는 swamping과 masking 문제의 위험이 줄어든다.

그림에서 파란 동그라미는 정상치를, 빨간 세모는 이상치를 나타낸다. 위의 그림을 통해 알 수 있는 비복원추출의 효과는 다음과 같다.

- 전체 데이터 (a)
    - 이상치 군집이 정상치 군집과 가까이 위치하고, 이상치 군집에 붙어있는 정상치들이 존재한다.
    - 이상치 군집이 정상치 군집에 비해 더 밀도가 높다.
- 비복원추출 후 샘플 (b)

- 이상치 군집과 정상치 군집이 분리되어 한눈에 보인다.
- 이상치 군집에 붙어있던 정상치들이 제거되었다.
- 이상치 군집의 크기가 줄어들어 이상치 탐지에 용이하다.

실제로도 전체 샘플을 이용했을 때 보다 비복원추출을 통해 얻은 샘플을 이용했을 때의 AUC가 더 높고, 이를 통해 비복원추출을 이용하면 swamping 및 masking 문제를 해결할 수 있음을 알 수 있다.

4. Anomaly Detection Using iForest

4.1 Training Stage

iForest의 첫번째 단계에서는 훈련 데이터를 비복원추출한 샘플을 통해 iTrees를 만들어낸다.

각 iTree들은 인스턴스들이 모두 isolated 상태가 되거나 트리의 일정 height에 도달할 때 까지 반복적으로 분할을 해낸다. 이 때, 트리의 height는 비복원추출의 크기에 의해 결정되고, 이는 트리들의 평균 height에 근사한다. 트리들의 평균 height이 기준이 되는 이유는 트리들의 평균 height보다 path length가 짧은 것들이 바로 이상치일 가능성이 높은 점들이기 때문이다.

구체적인 알고리즘은 다음과 같다.

Algorithm 1 : $iForest(X, t, \psi)$

Inputs: X - input data, t - number of trees, ψ - sub-sampling size
Output: a set of t *iTrees*

Initialize $Forest$
set height limit $l = ceiling(\log_2 \psi)$
for $i = 1$ to t **do**
  $X' \leftarrow sample(X, \psi)$
  $Forest \leftarrow Forest \cup iTree(X', 0, l)$
end for
return $Forest$

사진 출처: Fei Tony Liu et al., 2008

- Algorithm 1: iForest 함수

1. Forest 객체를 초기화한다.
2. 트리의 height 한계를 $l = ceiling(\log_2 \psi)$로 지정한다.
3. $i = 1$부터 트리의 개수 t까지 다음을 반복한다.
    a. 전체 데이터 X에서 비복원추출 크기 ψ만큼을 뽑아내어 이를 X'로 저장한다.
    b. 비복원추출한 데이터 X', 트리 개수 0, 트리 height 한계 l을 iTree 함수로 전달한다.
    c. iTree 함수의 결과를 Forest 객체에 추가한다.
4. Forest 객체를 반환한다.

Algorithm 2 : $iTree(X, e, l)$

Inputs: X - input data, e - current tree height, l - height limit
Output: an iTree

if $e \geq l$ or $|X| \leq 1$ **then**
  return *exNode*$\{Size \leftarrow |X|\}$
else
  let Q be a list of attributes in X
  randomly select an attribute $q \in Q$
  randomly select a split point p from max and min values of attribute q in X
  $X_l \leftarrow filter(X, q < p)$
  $X_r \leftarrow filter(X, q \geq p)$
  return *inNode*$\{Left \leftarrow iTree(X_l, e+1, l)$,
    $Right \leftarrow iTree(X_r, e+1, l)$,
    $SplitAtt \leftarrow q$,
    $SplitValue \leftarrow p\}$
end if

사진 출처: Fei Tony Liu et al., 2008

- Algorithm 2: iTree 함수
    1. 현재의 트리 height인 e가 트리 height 한계인 l보다 크거나 같은 경우, 또는 입력 데이터 X가 isolated 상태가 되어 데이터의 크기가 1보다 작거나 같은 경우,
        a. external 노드(말단 노드)로 반환한다.
    2. 그렇지 않은 경우,
        a. Q를 입력 데이터 X의 변수 리스트로 지정한다.
        b. 변수 리스트 Q의 원소 중 하나인 q를 랜덤으로 선택한다.
        c. 랜덤으로 선택한 변수 q에 대한 최댓값과 최솟값 사이에서 랜덤으로 분할점 p를 선택한다.

d. 변수 q에 대한 값이 p보다 작은 데이터들을 X_l(왼쪽)로 저장한다.

e. 변수 q에 대한 값이 p보다 크거나 같은 데이터들을 X_l(오른쪽)로 저장한다.

f. internal 노드들을 반환하고, 이 노드들에서 다시 위의 과정을 반복한다.

g. 현재의 트리 height인 e가 트리 height 한계인 l보다 크거나 같은 경우, 또는 입력 데이터 X가 isolated 상태가 되어 데이터의 크기가 1보다 작거나 같은 경우 트리 분할을 종료하며 external 노드를 반환한다.

iForest 함수에는 두개의 파라미터가 존재한다. 첫번째 파라미터는 비복원 추출의 크기를 나타내는 ψ이고, 이는 훈련 데이터의 크기를 조정한다. 경험적으로 ψ를 2^8로 설정하는 것이 좋은 성능을 내며, 이보다 크기를 키우는 것은 성능 저하, 과도한 실행 시간 및 메모리의 사용으로 이어지므로 크기를 키울 필요가 없다.

두번째 파라미터는 트리의 개수를 나타내는 t이고, 이는 앙상블 크기를 결정한다. 경험적으로 $t = 100$으로 설정하였을 때 path length가 잘 수렴하는 것으로 나타난다.

4.2 Evaluating Stage

이 단계에서는 테스트를 위한 인스턴스들이 iTree들을 통과하여 각 인스턴스별로 이상치 점수가 산출된다.

이상치 점수 s를 얻기 위해서는 각 인스턴스에 대한 path length의 기댓값인 $E(h(x))$를 알아야 한다. 해당 기댓값은 각 인스턴스를 iTree들에 넣어 각 트리에 대한 path length를 구함으로써 얻을 수 있다. 각 트리에 대한 path length를 구할 때는 아래와 같은 PathLength 함수를 이용하면 된다.

Algorithm 3 : $PathLength(x, T, e)$

Inputs : x - an instance, T - an iTree, e - current path length; to be initialized to zero when first called
Output: path length of x

1: **if** T is an external node **then**
2: return $e + c(T.size)$ {$c(.)$ is defined in Equation 1}
3: **end if**
4: $a \leftarrow T.splitAtt$
5: **if** $x_a < T.splitValue$ **then**
6: return $PathLength(x, T.left, e + 1)$
7: **else** $\{x_a \geq T.splitValue\}$
8: return $PathLength(x, T.right, e + 1)$
9: **end if**

사진 출처: Fei Tony Liu et al., 2008

- Algorithm 3: PathLength 함수
  1. 현재의 노드가 external 노드인 경우,
     a. 현재의 path length e에 adjustment에 해당하는 $c(T.size)$를 더한 값을 반환한다

        : $c(T.size)$를 더하는 이유는 트리 height 한계로 인하여 더 이상 분할이 진행되지 못하는 경우가 발생하기 때문이다. 즉, 현 노드에 여러 인스턴스가 존재하더라고 트리 height의 한계로 인해 더 이상 분할이 진행되지 못하는 것이다. 따라서 분할이 멈추지 않고 계속 진행됐을 때의 평균 path length인 $c(T.size)$를 adjustment로 더해주게 된다. 추가로 덧붙이자면 $T.size$는 현재 위치한 external 노드에 포함된 인스턴스 개수에 해당한다.
  2. 그렇지 않은 경우,
     a. 현재 노드에서 분할의 기준이 되는 변수(Algorithm 2에서의 q에 해당)를 a로 지정한다.
     b. 만약, 인스턴스 x의 변수 a에 대한 값이 분할점(Algorithm 2에서의 p에 해당)보다 작다면 왼쪽 트리로 이동하여 위의 과정을 반복한다.
     c. 만약, 인스턴스 x의 변수 a에 대한 값이 분할점(Algorithm 2에서의 p에 해당)보다 크거나 같다면 오른쪽 트리로 이동하여 위의 과정을 반복한다.

5. Empirical Evaluation

Section 1~4까지, 우리는 iForest(Isolation Forest)가 이상치를 분류하는 알고리즘에 대해서 짚어보았다. Section 5에서는 iForest가 다양한 실증적인 방법을 통해 실제 데이터에서 이상치를 탐지하는 성능을 평가해본다.

- 다른 이상치 탐지 모델들과 성능을 비교
- Sub-sampling size에 따른 효율성 분석
- 고차원 데이터에서의 IForest
- 이상치가 없이 학습을 진행한 경우 성능 분석

5.1 Comparison with ORCA, LOF, and Random Forest

해당 논문에서는 다양한 이상치 탐지 모델들과의 성능을 비교하였는데, 대표적으로 ORCA, LOF 그리고 RF 모델을 제시하였다. ORCA와 LOF 모델에 대한 간단한 설명은 아래 설명을 참고하자.

▼ **참고) About ORCA, LOF**

- **ORCA (Outlier detection and Robust Clustering for Attributed graphs)**

  ORCA는 iForest 이전에 등장했던 이상치 탐지의 방법 중 하나로, 다음과 같은 특징을 가지는 모델이다.

  1) k-Nearest Neighbour에 기반을 둔 이상치 탐지모델

  2) 이웃들간의 거리를 계산하여 이를 기반으로 이상치를 탐지하므로 Computing Intensity가 높은 방법

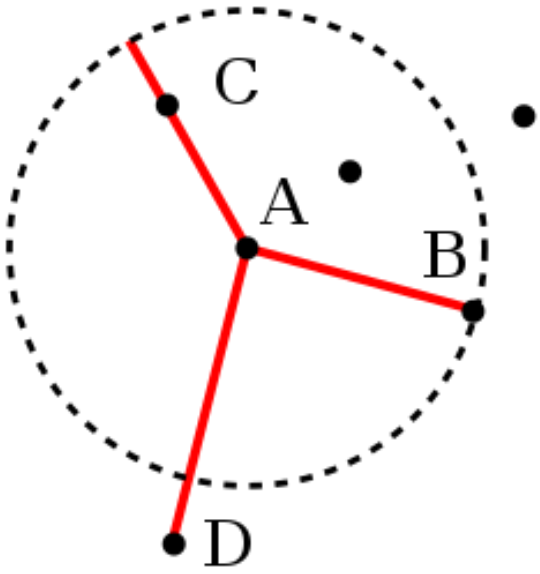
  3) random sampling을 함께 진행하여 complexity를 낮춤

  ORCA에서 조절해주는 파라미터 중 하나는 **k**로, k-nn 모델에 기반을 두었기 때문에 이웃들의 개수를 정해주는 파라미터이다. 일반적으로 k가 증가할 수록 이웃들간의 거리를 더 계산해야 하므로 시간이 더 오래 걸린다. 해당 논문에서는 ORCA 모델의 k에 대해 디폴트값으로 **k = 5**를 제시하였다.

  또 다른 파라미터는 **N**으로, 이상치 보고의 개수를 결정하는 파라미터이다. 보고된 이상치가 적다면, ORCA 모델은 이상치를 더 탐지하기 위해서 컷오프 값을 빠르게 증가시키고, 모델의 가지치기(pruning)를 더욱 증가시켜 이상치 후보군을 더 늘린다.

  하지만, 실제 보유하고 있는 이상치의 개수보다 큰 N을 설정하는 경우에는 anomaly score를 매길 때에 오류가 발생한다. 또한 학습 과정에서 실제 이상치의 개수는 알려져 있지 않다고 가정하기 때문에, 논문에서는 $N = \frac{n}{8}$로 제시한다. (n은 전체 instance의 개수)

- **LOF (Local Outlier Factor)**



k = 3일때의 LOF 알고리즘, D가 Anomal

LOF 또한 이상치 탐지의 방법 중 하나로, ORCA와 동일하게 k-nn 모델에 기반을 둔 모델이다.

ORCA와의 가장 큰 차이점은 모든 데이터를 전체적으로 고려하는 것이 아니라, 특정 관측치의 이웃을 이용하여 국소적(Local) 관점으로 이상치를 파악한다는 점이다. 파라미터는 ORCA와 동일하게 이웃의 개수인 **k**이다. 해당 논문에서는 k = 10을 이용하였다.

RF model은 ntree = 100으로 설정하였으며, 다른 파라미터들은 디폴트 값으로 설정되었다.

본격적인 모델 비교에 앞서, RF는 지도학습의 한 형태이므로 정답이 제시된 학습 데이터가 필요하다. 따라서 해당 논문에서는 randomly, uniformly sampling을 진행하여 대립적인 class가 존재하는 합성 데이터를 제작하였다. (Mulcross Data, 이상치 비율 = 10%, distance factor = 2, anomaly clusters = 2)

총 11개의 데이터셋 + 1개의 합성 데이터로 분석을 진행하였고, 모든 명목형과 이진 변수는 제거된 채로 테스트를 진행하였다. 기본적으로 Isolation Forest는 수치형 자료에서 이상치를 탐지하는데 적합한 방법이다.

- 명목형 변수에 Isolation Forest를 적용하려면 label Encoding 등의 방법을 적용해서 진행해주어야 한다.
- 만약 명목형 변수가 Ordinal하다면, Ordinal하다는 전제 하에 Ordinal Encoder를 사용해주고, 그렇지 않다면 One-Hot encoding 등의 방법을 사용하는 것이 일반적

으로 효과가 좋다.

	AUC				Time (seconds)					
	iForest	ORCA	LOF	RF	iForest			ORCA	LOF	RF
					Train	Eval.	Total			
Http (KDDCUP99)	**1.00**	0.36	NA	NA	0.25	15.33	**15.58**	9487.47	NA	NA
ForestCover	**0.88**	0.83	NA	NA	0.76	15.57	**16.33**	6995.17	NA	NA
Mulcross	**0.97**	0.33	NA	NA	0.26	12.26	**12.52**	2512.20	NA	NA
Smtp (KDDCUP99)	**0.88**	0.80	NA	NA	0.14	2.58	**2.72**	267.45	NA	NA
Shuttle	**1.00**	0.60	0.55	NA	0.30	2.83	**3.13**	156.66	7489.74	NA
Mammography	**0.86**	0.77	0.67	NA	0.16	0.50	**0.66**	4.49	14647.00	NA
Annthyroid	**0.82**	0.68	0.72	NA	0.15	0.36	**0.51**	2.32	72.02	NA
Satellite	**0.71**	0.65	0.52	NA	0.46	1.17	**1.63**	8.51	217.39	NA
Pima	0.67	**0.71**	0.49	0.65	0.17	0.11	0.28	**0.06**	1.14	4.98
Breastw	**0.99**	0.98	0.37	0.97	0.17	0.11	0.28	**0.04**	1.77	3.10
Arrhythmia	**0.80**	0.78	0.73	0.60	2.12	0.86	2.98	**0.49**	6.35	2.32
Ionosphere	0.85	**0.92**	0.89	0.85	0.33	0.15	0.48	**0.04**	0.64	0.83

사진 출처: Fei Tony Liu et al., 2008
4가지의 모델과 성능 비교표, 일반적으로 iForest가 AUC가 높고, 특히 Big Data에 대해 속도가 빠르다.

결과표를 보면 알 수 있듯이, iForest는 특히 large data(Http, Smtp, Mulcross)에 대해서 다른 모델들에 비해 AUC가 높고, Process time이 매우 짧다는 장점이 있다. 이런 결과를 보이는 데에는 다음과 같은 이유가 존재한다.

- ORCA, LOF 모델은 모두 k-nn 모델을 기반으로 하므로, 거대한 데이터셋의 개별 관측값 간 거리를 다 계산해주는 과정이 필요하다. 하지만 iForest 모델은 거리기반 알고리즘을 이용하지 않기 때문에 해당 과정이 생략되므로 두 모델에 비해 매우 빠른 속도를 보여준다.
- 또한 ORCA 모델은 anomaly cluster의 개수가 dense한 Http와 같은 데이터셋에 대해서는 낮은 AUC를 보인다. 해당 문제는 k의 값을 증가시키면 해결할 수 있지만, 처리 시간이 증가하기 때문에 별로 실용적이지는 못하다.

5.2 Efficiency Analysis

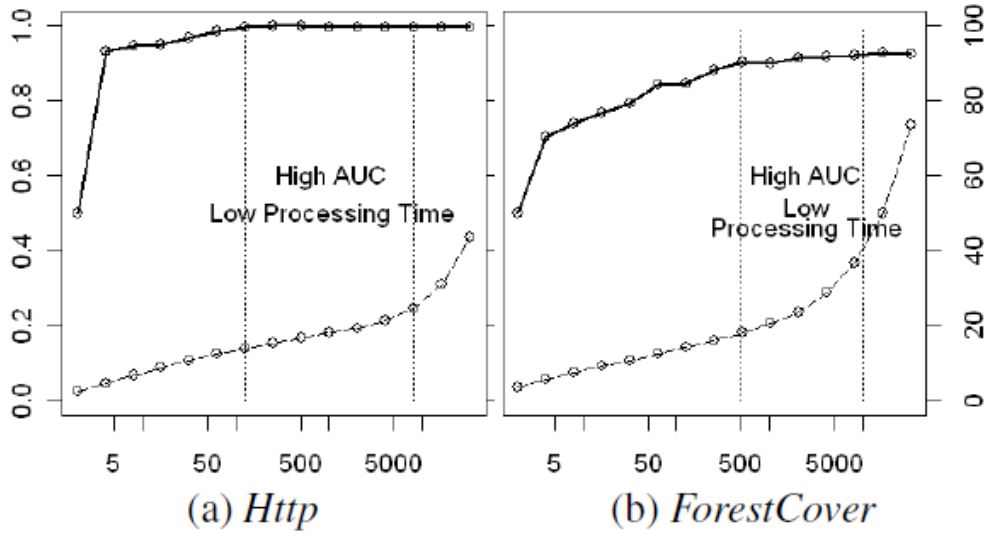


사진 출처: Fei Tony Liu et al., 2008

iForest의 효율성은 sub-sampling size ψ와 연관이 있다. 따라서 2개의 large dataset을 통해 ψ값이 변함에 따라서 AUC값과 처리 시간의 관계를 파악해본다. Test에서 사용한 파라미터는 $\psi = 2, 4, 8, 16, ..., 32768$이다.

논문의 앞에서 제시했듯이, 실험 결과 일반적으로 성능이 좋은 값은 $\psi = 128, 256$ 등이다. 해당 값들보다 sub-sampling size를 키우게 되면, AUC 값은 modest하게 증가하는 대신 processing time은 기하급수적으로 증가하므로 Elbow point라고 할 수 있는 지점에서 선택하는 것이 적절해보인다. (a) 데이터셋의 경우는 $\psi = 128$, (b) 데이터셋의 경우는 $\psi = 512$가 가장 최적의 값으로 선택되었다.

5.3 High Dimensional Data

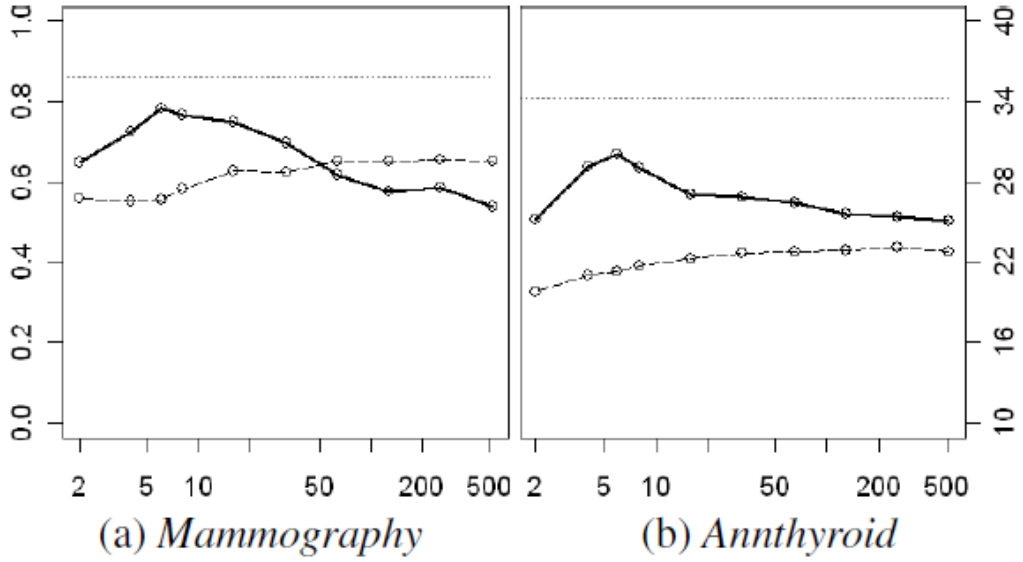


사진 출처: Fei Tony Liu et al., 2008

해당 논문에서는 고차원 데이터 문제를 다루기 위해서 Original Dataset인 Mammography, Annthyroid에 약 506개의 noise 변수를 추가하여 테스트를 진행한다.

Anomaly detection에서 다른 모델들이 차원의 저주 문제로 좋지 못한 성능을 보이는 것과 같이, iForest도 동일한 문제가 발생한다. 해당 논문에서는 이런 문제를 보완해주기 위해 iForest를 적용하기 전에, 첨도 검정을 통해서 sub-sample로부터 변수들의 subspace를 먼저 선정해주었다.

- 첨도는 이상치의 존재에 상당히 민감한 통계량이므로, 이상치 탐지에 대해 좋은 변수 선택 방법이 될 수 있다. 개별 변수들의 첨도를 계산한 뒤 Ranking 순으로 정렬을 하여 각각의 트리에 맞는 변수 공간을 선정한다.

해당 논문에서는 subspace size가 noise 변수를 제외한 변수들의 원래 개수와 동일할 때 상승한다는 것을 확인하였다. (약 6개)

그 외에도 이상치 탐지를 위한 Grubb's Test와 같은 방법들도 논문에서 간단히 언급하였다. 아래 내용은 참고용이다.

▼ **참고 : Grubb's test for anomaly detection**

- 선행 조건 : Dataset이 Gaussian Distribution이며, 적어도 7개의 data가 존재해야 함

- 방법론 요약

  1) ESD 검정 실시

  가) 최댓값이 이상치로 의심될 경우

  $G = \frac{max(x_i)-\bar{x}}{s}$, $\bar{x}$: 표본평균, s : 표본표준편차

  나) 최솟값이 이상치로 의심될 경우

  $G = \frac{\bar{x}-min(x_i)}{s}$

  다) 긴가민가 할 경우

  $G = \frac{max|x_i-\bar{x}|}{s}$

  2) 기각역 구해서 검정 실시

  $G_{critical} = \frac{(n-1)}{\sqrt{n}}\sqrt{\frac{(t_{a/(2N),N-2})^2}{N-2+(t_{a/(2N),N-2})^2}}$

위에서 제시한 첨도 검정 외에도 비슷한 변수의 특성을 찾아 변수를 줄여주는 방법을 활용하면 iForest의 성능이 높아진다. iForest는 ORCA 등 다른 모델과 다르게 변수가 많은 고차원 데이터에서도 빠른 성능을 보여준다는 강점이 있다.

5.4 Training Using Normal Instances Only

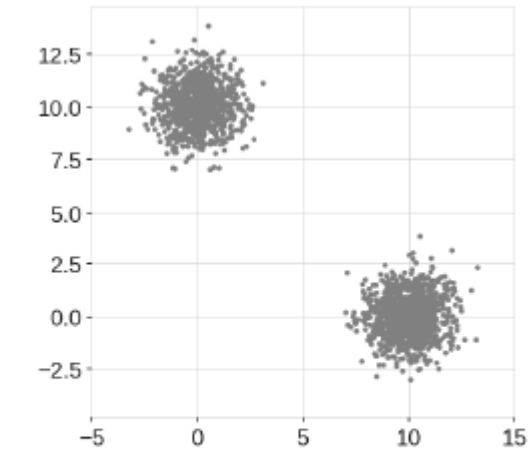
iForest가 학습을 진행할 때, normal instance로만 학습을 진행하여도 large Dataset에 대해 AUC 성능이 각각 0.0078, 0.0085 정도로만 감소하였다.

비록 AUC 감소량이 적지만, sub-sampling size를 증가시켰을 때 다시 AUC가 증가한 결과가 나왔다. 이러한 방법을 통해 편중된 학습을 통한 AUC 감소 문제를 해결할 수 있다.

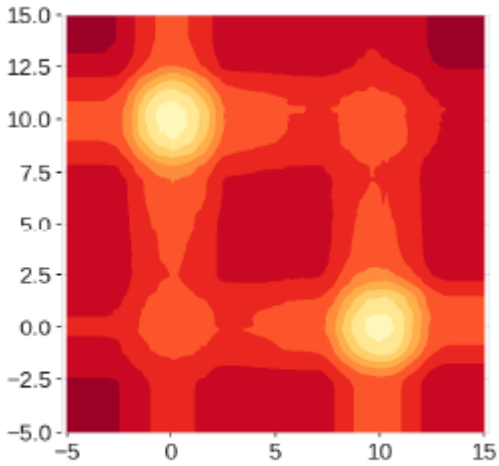
Appendix. Extended Isolation Forest

> Hariri et al. (2018)

- 위에서 서술한 Isolation Forest를 통해 우리는 효율적으로 이상치를 탐지하는 방법들에 대해서 알아보았다. 그렇다면 Isolation Forest는 항상 잘 작동하는 방식이며 모든 데이터셋에 적합할까? 그것은 아니다. 따라서 해당 부분에서는 Isolation Forest의 확장판이라고 할 수 있는 Extended Isolation Forest에 대해서 간단하게 다루어보자.

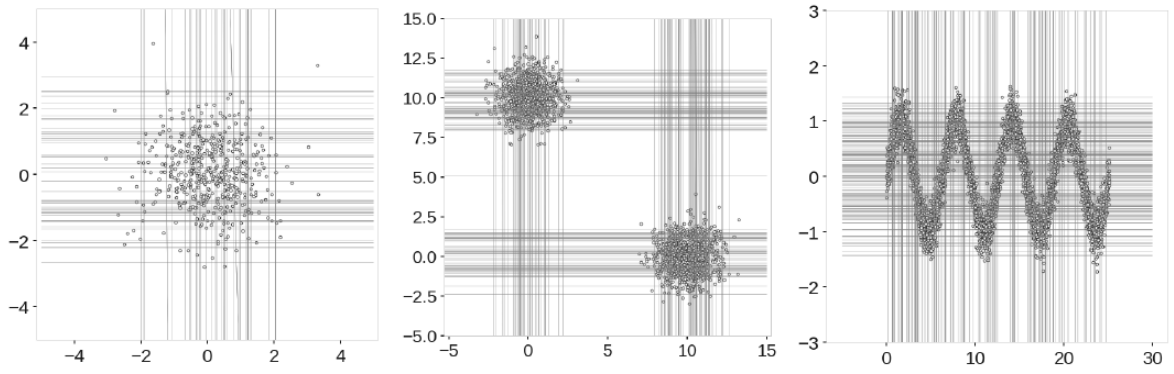


(a) Two normally distributed clusters



(b) Anomaly Score Map

- 다음과 같이 분포하는 데이터셋을 생각해보자. 일반적으로 의사결정나무는 가로, 세로 축과 수직 또는 수평으로 분할을 진행해나간다. 그림에서 밝은 색으로 보이는 부분은 정상치에 가까운 부분, 어두운 부분은 이상치에 가까운 부분이라고 할 수 있다.
- 실제 데이터를 보면 1시와 7시 방향에 데이터가 위치하면 정상값에서 벗어난 이상치 데이터라고 볼 수 있다. 하지만 실제로 iForest 방법을 통해 Anomaly Score를 계산해보면 오른쪽 그림과 같이 어느 정도 정상치라고 생각할 수도 있게 된다.

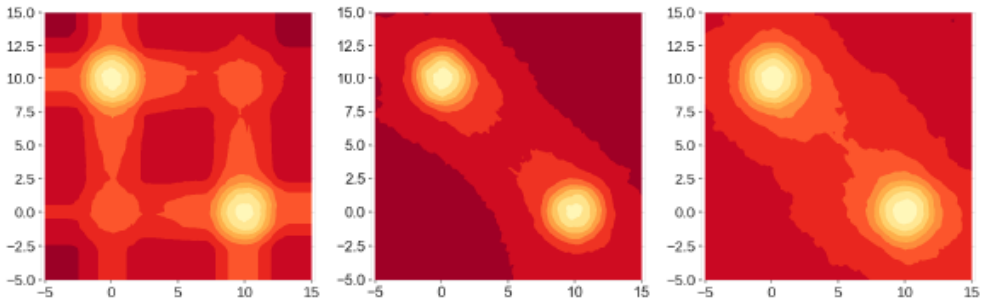


(a) Single blob (b) Multiple Blobs (c) Sinusoidal

데이터가 (b)와 (c) 그래프와 같은 경우 이런 문제가 종종 발생한다.

- 이런 현상이 발생하는 이유는 1시와 7시 부분에 위치한 데이터들은 Normal data들에 대해 판단을 진행하는 과정에서 한쪽 변수는 이상치로, 한쪽 변수는 정상치로 판단이 되기 때문이다. 따라서 Anomaly score 상에서 다른 이상치들에 비해 상대적으로 정상인 것처럼 보이게 되는 것이다.

- Extended Isolation Forest에서는 기존의 Isolation Forest 알고리즘에서 의사결정나무의 분할 시 기울기와 절편을 랜덤하게 부여하는 방식으로 해당 문제를 해결하게 된다.
- 기존의 iForest 알고리즘에서 4번, 5번 항목이 추가되었는데, 각각 의사결정나무 분할 이전 기울기와 절편이 변하면서 위의 그림과 같이 불규칙적인 분할이 이루어지게 되고, 이를 통해서 이상치를 효과적으로 탐지할 수 있다.



(a) Standard IF (b) Rotated IF (c) Extended IF

- Extended IF 알고리즘을 적용하여 우리는 효과적으로 패턴이 있는 데이터에 대해서도 이상치를 탐지할 수 있다. 따라서 데이터의 특성과 형태를 잘 파악하여 IF 모델과 Extended IF 모델을 적절히 활용하는 것이 필요하다.
    - 참고 : Rotated IF는 데이터의 정보(분산)가 가장 많이 존재하는 축을 기반으로 데이터를 회전시켜 IF를 적용하는 방식이다. PCA와 유사하다고 생각하면 이해하기가 편할 것 같다.